LA-UR-02-6987

Title: Simulating the Quadrics Interconnection Network

Author(s): Kathryn Berkbigler, Graham Booker,
Brian Bush, Kei Davis, and Nicholas Moss

Submitted to: High Performance Computing Symposium 2003
Advance Simulation Technologies Conference 2003
30 March-3 April 2003
Orlando, Florida

Simulating the Quadrics Interconnection Network

**Kathryn Berkbigler, Graham Booker,
Brian Bush, Kei Davis, and Nicholas Moss
Los Alamos National Laboratory
Los Alamos, NM 87545
{kpb,gbooker,bwb,kei,nickm}@lanl.gov**

Keywords: Quadrics network, discrete event simulation, direct execution, architecture, validation.

ABSTRACT

We outline *à la carte*, an approach for simulating computing architectures applicable to extreme-scale systems (thousands of processors) and to advanced, novel architectural configurations, and describe in detail our simulation model of the Quadrics interconnection network. Our component-based design allows for the seamless assembly of architectures from representations of workload, processor, network interface, switches, etc., with disparate resolutions and fidelities, into an integrated simulation model. This accommodates different case studies that may require different levels of fidelity in various parts of a system. Simple ping timings can be modeled to approximately 100 ns. We present results comparing the simulated versus actual execution time of a 3D neutron transport application run on a machine with a Quadrics network.

INTRODUCTION

The magnitude of the scientific computations targeted by the US DOE ASCI project requires unprecedented computational power, and bandwidth to enable remote, real-time interaction with the compute servers. To facilitate these computations ASCI plans to deploy massive computing platforms, possibly consisting of tens of thousands of processors, capable of achieving 10-100 TeraOPS, with WAN connectivity from these to distant sites.

The *à la carte* project seeks to develop a simulation-based analysis tool for evaluating massively parallel computing platforms including current and future ASCI-scale systems. This tool will provide a means to analyze and optimize the current systems and applications as well as influence the design and development of next-generation high-performance computers. Hence our general goal is to design and implement a *flexible* and *modular* simulation framework for design and analysis of extreme-scale parallel and distributed computing systems, and as an ongoing part of this process to validate the accuracy of results produced by any particular model.

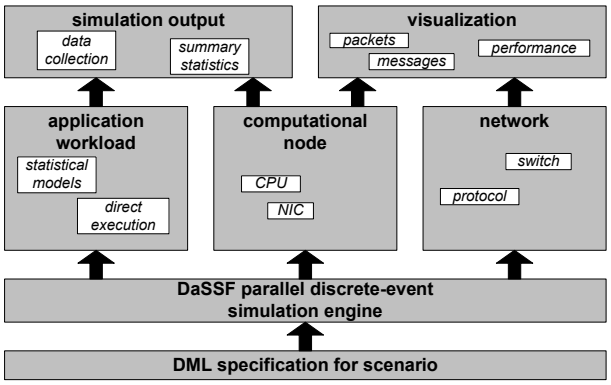

Figure 1: The architecture of the *à la carte* simulator.

Our basic approach relies on an iterative development process for constructing components of appropriate fidelities and integrating them into a portable and efficient parallel discrete-event simulation that is scalable to thousands of (simulated) computational nodes. Components may be processors, switches, network interfaces, or application workloads, for example. Studies of hardware architectures are made by running our simulation for a particular aggregate system composed of these components. The output of the simulation captures the behavior and performance of the components, and may be visualized using the *à la carte* visualizer. (The visualizer utilizes the Flatland framework created at the University of New Mexico as part of the Homunculus project [1].) Figure 1 illustrates the architecture of our simulator.

We chose a portable, conservative synchronization engine, the Dartmouth Scalable Simulation Framework (DaSSF) [2, 3], Dartmouth College's implementation of the Scalable Simulation Framework API [4], for the handling of discrete events. DaSSF manages the synchronization, scheduling, and delivery of events in the simulation; it has a lean C++ API and supports both shared-memory and distributed-memory parallelism. The SSF

API provides five base classes that applications may subclass. The **SSF_Entity** class defines the entities in the simulation and maintains their state information. The **SSF_Process** class defines the behaviors that entities possess. Entities are connected to each other via channels, an **SSF_OutChannel** in the transmitting entity and an **SSF_InChannel** in the receiving entity. An **SSF_Event** represents the information that flows between entities across the channels. Additionally, DaSSF enhances SSF by providing classes for semaphores, timers, random number generation, data collection, and simple statistics. DaSSF is available for a variety of platforms.

We use the Domain Modeling Language (DML) [2] to specify the architecture and workload to be simulated. Properties of model components, the number to be instantiated, and their connectivity are specified in the model DML file. Another DML file contains runtime information such as simulation start and end times and the values of global parameters. DaSSF provides a partitioner that constructs the simulation components from the topology in the model DML file and distributes these components over the parallel computing platform. The lower two levels of the architecture in Figure 1 comprise DML and DaSSF.

The component-based design (represented by the middle layer in Figure 1) allows for the seamless assembly of architectures from representations of workloads, processors, network interfaces, switches, etc., with disparate resolutions, into an integrated simulation model. One can mix and match components of different fidelities to construct a model with the appropriate level of detail for a particular study. We are focusing on the development of a simulation capability that scales to tens of thousands of processors and that can execute on a wide variety of computing platforms which may themselves be very large.

The current *à la carte* implementation comprises low- and medium-fidelity models of a network and low-fidelity and direct-execution models of workload. The next section briefly describes our current workload models. Our medium-fidelity representation of the Quadrics interconnection network follows. This implementation supports studies of simulation performance and scaling, and also the properties of the simulated systems themselves. Ongoing work aims to improve the fidelity of the representations and protocols with validation at each stage. Future work will further emphasize the representation of I/O and storage, and wide-area networking.

REPRESENTING APPLICATION WORKLOADS

In the simplest workload model, the user specifies exactly what messages will be sent from each SMP node in the network topology, the time the message is sent, the destination of the message, the message size in bytes, and optionally, the data content of the message. This workload model is useful for testing specific features of the network models because the content of messages is precisely controlled. It can also be used in trace-driven studies of network behavior.

The statistical model for a workload is characterized by three random variables: an exponentially distributed delay between messages, an exponentially distributed message size, and the message destination, where all possible destinations are equally likely. The average values for message delay and size are specified in the DML model input. The destinations to which each source node can send can be specified individually for each node.

The ping workload model was developed to facilitate comparison of the simulation with ping tests conducted on the network hardware. Parameters for this workload are the exact size of the message, the exact delay between messages, the number of messages to send, and the message destination for each message source.

The direct execution workload component provides a means to generate network messages according to the demands of an actual running application. In direct execution simulation the application is executed on the same machine used to perform the simulation. The application is typically modified to call the simulator only for those operations that differ between the host machine and the simulated machine. Using the host machine to directly execute some instructions rather than simulating all instructions can result in considerably faster execution with minimal loss of accuracy when the host and target have similar architectures. The experiments with direct execution thus far have focused on simulation of communications on the interconnection network and direct execution of the computational aspects of an application.

QUADRICS INTERCONNECTION NETWORK MODEL

The primary requirement is the ability to accurately model the movement of packets in Quadrics networks consisting of Elan network interface cards [5, 6] connected to Elite crossbar switches [7, 6] in a fat-tree network at nearly *flit* (16-bit unit) resolution. Rather than model the processors and memory hierarchy of an SMP node in any detail the emphasis is on modeling the interconnection network and routing protocol. We need to accurately track the movement of the message across the PCI bus between main memory and the network interface card (NIC), account for its packetization, and clock the transfer of data across the network. Because contention may exist in the network, different parts of the packet may move at different speeds through the switches (i.e., buffering and delays may occur anywhere in the network).

The model contains three types of DaSSF entities, representing the SMP node, the NIC, and the network switch. The SMP node has an outgoing channel for sending messages to its NIC and an incoming channel for receiving messages from its NIC. The NIC has a corresponding incoming channel for receiving messages from its SMP node and an outgoing channel for sending messages to its SMP node. Additionally, the NIC has an outgoing channel and an incoming channel that connect it to its network switch. Each network switch has 8 incoming channels and 8 outgoing channels. At the level nearest the NICs, 4 of the channels communicate with 4 NICs and 4 communicate with the next level in the quaternary fat-tree. Higher in the fat-tree, communication involves only other switches. Figure 2 illustrates the layout of such a network with 64 computational nodes and three layers of 16 switches each.

DaSSF entities are assigned to timelines in the model DML file. A DaSSF *timeline* (another name for a logical process) is a submodel that may run concurrently with other submodels [3]. Entities on different timelines communicate exclusively through messages passed over channels. The assignment of entities to timelines may have a large effect on simulation performance, especially when the latencies on the channels differ greatly.

The SMP node has two DaSSF processes, **TWorkloadSender** for sending messages and **TWorkloadReceiver** for receiving messages. The route that a message takes through the network is determined at the source. The NIC has a routing algorithm which is composed of four tightly coupled processes: **TWorkloadListener** receives messages from the source SMP and buffers them if the NIC is busy, **TBusTransfer** models the transfer of data across the PCI bus which is represented by a DaSSF semaphore, **TNICSender** splits the message into packets and sends the packets to the network switch, and **TNICReceiver** receives packets from the switch and sends the completed message to the destination SMP. The network switch has a flow control algorithm which has one process, **TCrossbarSwitch**, that receives packets on an incoming channel from a NIC or another switch and forwards them out the appropriate outgoing channel to the next switch or NIC as specified by the route that is embedded in the packet. DaSSF timers are used to control the interleaving of operations related to the simultaneous passage of multiple packets through a switch. DaSSF events are defined for messages and for the packets in a message. The precise behavior that occurs in the processes depends on the type of event that arrives.

The design relies on the tracking of the head and tail of the packet throughout its history, and also of various flit-level tokens specified in the Elan protocol. Figure 3 illustrates the sequence of operations it takes to move a message from one computational node to another through this sort of network. The existence of two virtual channels sharing bandwidth at switches (but without age-based priorities, etc.) is accounted for. The Elan *ACKNow* request may occur anywhere within the packet (usually after 64 bytes or at the end of the packet). The *EOP_GOOD* tokens free the virtual channels used by the packet. The *START/STOP* tokens are accounted for by buffering of packets at the incoming links to switches if no outgoing virtual channel is available. The PCI bus is modeled as half-duplex, and accounts for writes to the NIC's command port. Finally, wildcards are allowed for packet routing on upward links. Error conditions or the hardware support for broadcast communications are not yet modeled. We have determined that adding further resolution to this network model may not be cost effective because of the uncertainties involved in the performance of the operating system on the node, memory issues, and PCI bus behavior.

The basic strategy for dealing with packets is as follows. When the head of the packet reaches an entity like a NIC, switch, or node, it leaves a reservation at the entity. The head of the packet is forwarded along the route as soon as possible—it might be delayed slightly for switch logic or may be delayed significantly if it is queued for later transmission. As soon as the head leaves the entity, the reservation keeps track of how many bytes remain to be transmitted. The tail of the packet cannot be forwarded until it has been received from the previous stage and the number of bytes remaining at the current stage is zero. The "okay" event proceeds along the reverse path at full speed, and the "good" event cleans up the reservations. There is some fairly complex timekeeping logic for multiplexing the transmission of packets in switches and the receipt of them in NICs.

We track the leading edge ("head"), trailing edge ("tail"), *ACKNow* request, *PACK_OK* token, and *EOP_GOOD* token for packets in the network, which are implemented as DaSSF events.

Scaling Behavior

In order to understand how our Quadrics network simulation scales as a function of the size of the machine on which it is run, we have performed direct execution simulations of the ASCI SWEEP3D application, which solves a "one-group, time-independent, 3D Cartesian discrete ordinates neutron transport problem" [8]. Three partitions of a single SWEEP3D problem involving a $50 \times 50 \times 50$ cell computational grid were studied: a 2×2 partition of computational nodes (with one node per CPU), a 3×4 partition, and a 6×6 partition. For each of these we varied the number of computational nodes (with one node per CPU again) used for the simulator and measured its performance. All simulations were run on the *wolverine* machine (*wms.acl.lanl.gov*), a 64-node

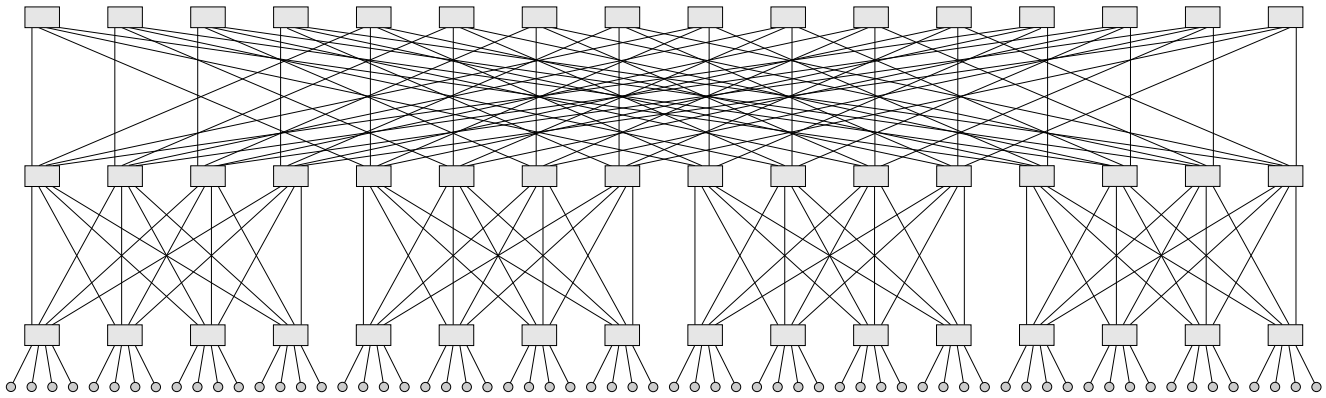

Figure 2: Quaternary fat-tree network with 64 computational nodes: the circles represent SMP nodes, the rectangles represent switches, and the lines represent cables.

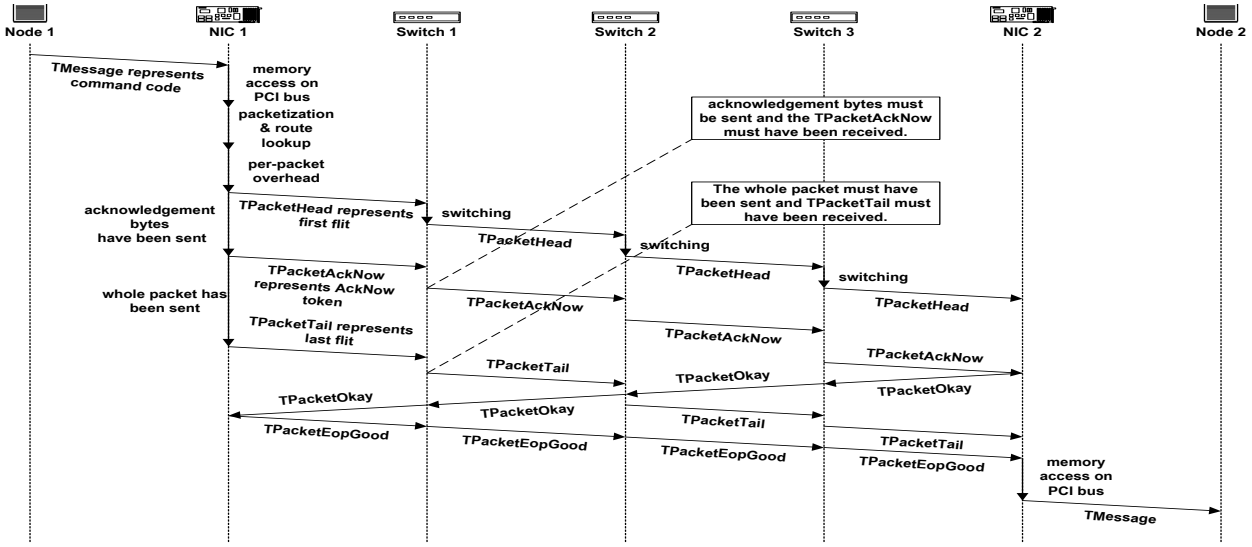

Figure 3: Steps in the simulated movement of a message from one computational node to another in a Quadrics network of Elan network interface cards and Elite switches.

Compaq machine with four ES-40 CPUs per node and a three-layer Quadrics network (see Fig. 2).

Because there are large time delays between the initiation of a message send on a node and its actual departure from a NIC, it is most efficient to partition the simulation so that the network and NICs reside on the same DaSSF timeline (since they have short delays between each other). The nodes are evenly distributed among the remaining timelines. For all of the simulation runs we assign a single timeline to each computational node (CPU).

Figures 4 through 7 illustrate the scaling behavior of our simulation of this application. It is clear that the addition of computational nodes does not reduce the overall execution time (Fig. 4) or average CPU time (Fig. 5). (Note that the jitter in the timing results in Figs. 4 and 5 results from having only single measurements for each data point—which are susceptible to the competition for machine resources, operating system behavior, etc.—rather than an ensemble of measurements whose average or minimum could be taken.) We expect reductions in time to be apparent when much larger SWEEP3D problems are simulated, however. The peak memory usage (Fig. 6) of the simulation increases as a function of the number of simulation nodes because event queues tend to be larger when more timelines are present. Similarly, more kernel communication is necessary (Fig. 7) when more timelines are present. The fact that we can simulate a 36-process run of SWEEP3D on two processors in a reasonable amount of clock time (left side of Fig. 4) and that the simulation performance does not degrade when it is distributed (right side of Fig. 4) indicates that we will be able to simulate SWEEP3D problems that are at least an order of magnitude larger than the ones presented here.

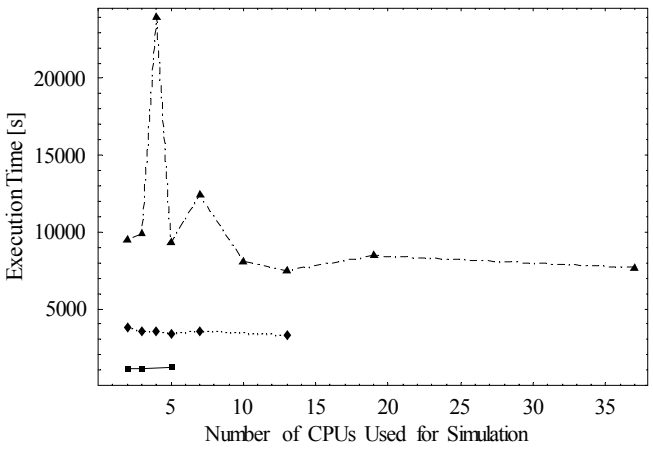

Figure 4: Execution time of the simulator (not the simulated application) as a function of the number of nodes (CPUs) on which it was run for three different partitionings of a SWEEP3D problem: 2×2 (squares), 3×4 (diamonds), and 6×6 (triangles).

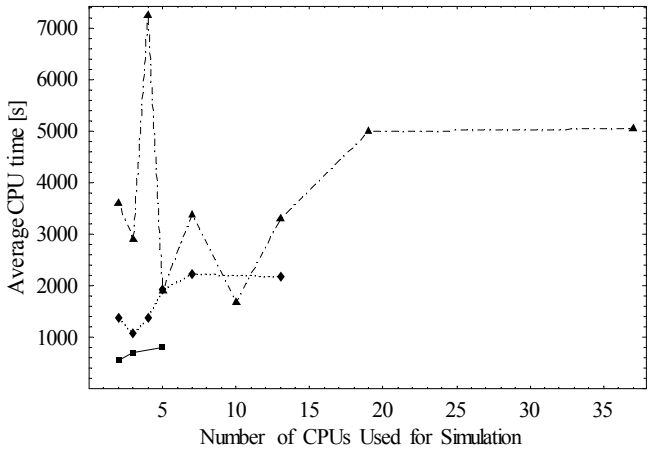

Figure 5: Average CPU time (i.e., total CPU time divided by the number of CPUs) used by the simulator (not the simulated application) as a function of the number of nodes (CPUs) on which it was run for three different partitionings of a SWEEP3D problem: 2×2 (squares), 3×4 (diamonds), and 6×6 (triangles).

Network Calibration

Before attempting to validate the Quadrics simulation against a real application we need to determine the various parameters used by the simulation. Many of these can be set to values given in the network design specifications, but some must be found empirically. To do this we consider the case of "pinging" one node with a message from another node. We examine three types of pings: (i) sending a message from the onboard memory of an Elan network interface card to another; (ii) sending a message from the main memory of a node to another; and, (iii) sending a message using the MPI protocol, which may use additional buffers in main memory. For each of these cases we performed a large set of pings using different message sizes and passing through different portions of the interconnection network.

Table 1 shows the parameters that must be set in any simulation: the ones shown in boldface type were set by

location	parameter	Elan-Elan	Main-Main	MPI
NIC	message delay	**3.000 μs**	**3.750 μs**	**11.000 μs**
NIC	packet delay	**120 ns**		
NIC	packet size	320 b		
NIC	acknowledgement bytes	at end		
NIC	bus bandwidth	∞	**800 MB/s**	**950 MB/s**
NIC, Switch	network bandwidth	400 MB/s		
Switch	packet delay	36 ns		
Node to NIC	channel delay	(bus bandwidth)$^{-1}$		
NIC to Level 1 Switch	channel delay	**3 ns**		
Level 1 Switch to Level 2 Switch	channel delay	1 ns		
Level 2 Switch to Level 3 Switch	channel delay	**10 ns**		

Table 1: Parameters used in ping simulations: the parameters in boldface type were set from measurements, rather than from design specifications.

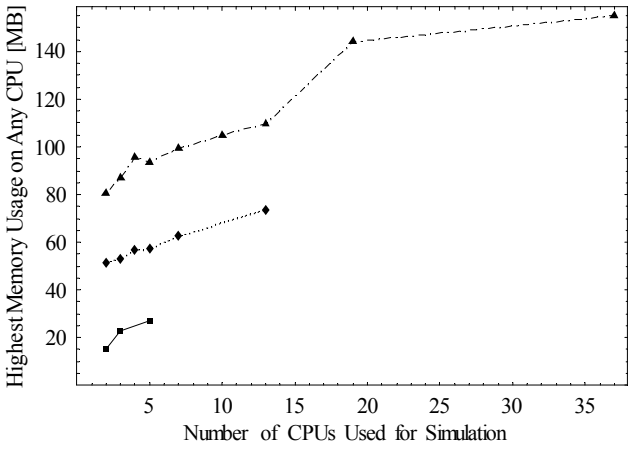

Figure 6: Peak memory usage on any CPU for the simulator (not the simulated application) as a function of the number of nodes (CPUs) on which it was run for three different partitionings of a SWEEP3D problem: 2×2 (squares), 3×4 (diamonds), and 6×6 (triangles).

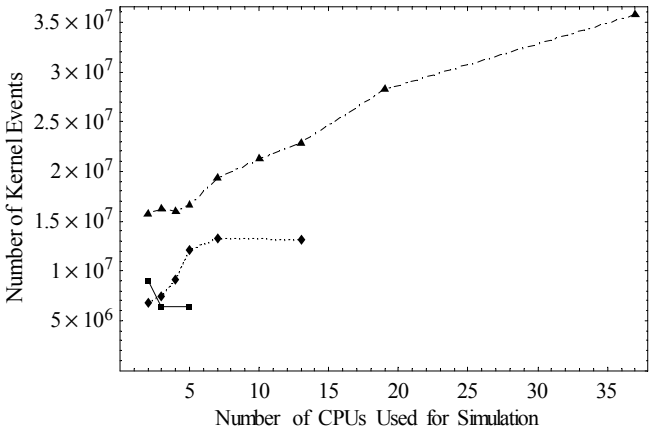

Figure 7: Number of simulation kernel events in the simulator (not the simulated application) as a function of the number of nodes (CPUs) on which it was run for three different partitionings of a SWEEP3D problem: 2×2 (squares), 3×4 (diamonds), and 6×6 (triangles).

studying the results of regression analysis of ping measurements; the other parameters were determined from design documentation describing the Elan/Elite hardware. Figure 8 shows the raw data. We encountered several significant difficulties in determining these parameters, most of these related to issues on which the Elan/Elite documentation [5, 7] provided little guidance as to the detailed behavior of the hardware. There was, for instance, an anomalous discontinuity in measured latencies for messages involving one versus two packets if the packets have to pass above the first switch layer in the network. The slopes of all three series of data (Fig. 8) seem inconsistent with the implication in the Elan/Elite documentation that subsequent packets in a message incur the same switch delay as the first packet. Also, the fact that the triangles in Fig. 8 form series with two different slopes suggests that there may be a hardware problem on this particular machine introducing additional latency in part of the third level of switches. It also appears that packets are delayed much more between switch layers 2 and 3 than between 1 and 2, even though a short wire connects adjacent layers—they all reside in the same chassis.

Figures 9–11 show the quality of agreement of our ping simulations with measured data. For Elan-memory pings, we achieve about 100 ns accuracy; for main-memory and MPI pings, we achieve about 750 ns accuracy. The main-memory and MPI ping accuracy can probably only be improved by incorporating higher resolution memory and bus models into our simulation.

ASCI SWEEP3D VALIDATION

In order to validate our Quadrics simulation, we have compared the actual running time of the SWEEP3D application with our simulation of it using the direct-execution workload. We consider the same three SWEEP3D partitions discussed in the section on scaling behavior. SWEEP3D and all of the simulations were run on *wolverine*; some of the simulations were run when this machine was loaded with other applications, so our measurements contain noise and systematic errors. A kernel patch supporting high-resolution timers was unavailable at the time so the standard Linux timing functions, which have a resolution of 1 ms, were used. (Times measured as zero were rounded up to 50 μs, based on an analysis of execution traces of SWEEP3D.) These experiments will be rerun when a high-resolution timer is available and when the machine is unloaded.

Figure 12 shows that our simulated execution time generally agrees with the actual execution time to within 10%. The couple of points with larger errors may have resulted from interference with other jobs running on the machine during the simulation. Note that no parameters in our network model were adjusted for this validation—all of the parameters were determined purely from the MPI ping analysis above. The simulated program produces numerical results for the neutron transport calculation that are identical with those obtained by running the program itself.

CONCLUSION

We have outlined the design and implementation of our Quadrics network model and our workload representations along with the simulation technology that supports them. Our component-based development process enables seamless composition of hardware, protocols, and workloads of varying fidelities into a single simulation. These models have been calibrated to the behavior of an existing cluster computer of 64 nodes with 256 Alpha/Linux processors connected by an ELAN3 Quadrics network. The calibrated simulation accurately represents real MPI-based pings on the network to within about 750 ns; lower-level pings are modeled accurately to about 100 ns. The network and workload modules have been validated against the real behavior of a representative ASCI application, SWEEP3D: we can predict the execution time of this application to within about 10%, even using the relatively coarse application timers available for our experiments. Our study of the scaling properties of our simulation indicates that it can handle much larger application instances than SWEEP3D.

ACKNOWLEDGEMENTS

This work was carried out under the auspices of the Department of Energy at Los Alamos National Laboratory under ASCI DisCom2. We would like to thank LANL's DisCom project leader, Stephen Turpin, and the *à la carte* project leader, Adolfy Hoisie, for their support.

Steve Smith (at LANL) and Thomas P. Caudell, Kenneth L. Summers, and Cheng Zhou (at the University of New Mexico's Albuquerque High Performance Computing Center) led the development of the visualization software used by *à la carte*.

Thanks to members of the extended LANL *à la carte* team for technical input, including Mike Boorman, Fabrizio Petrini, and Harvey Wasserman; and to David Nicol and Jason Liu (Dartmouth College).

REFERENCES

[1] `http://www.ahpcc.unm.edu/homunculus/indexold.html`.

[2] Liu, J. and D.M. Nicol. 2002. "Dartmouth Scalable Simulation Framework User's Manual." Dept. of Computer Science, Dartmouth College, Hanover, NH, February 6.

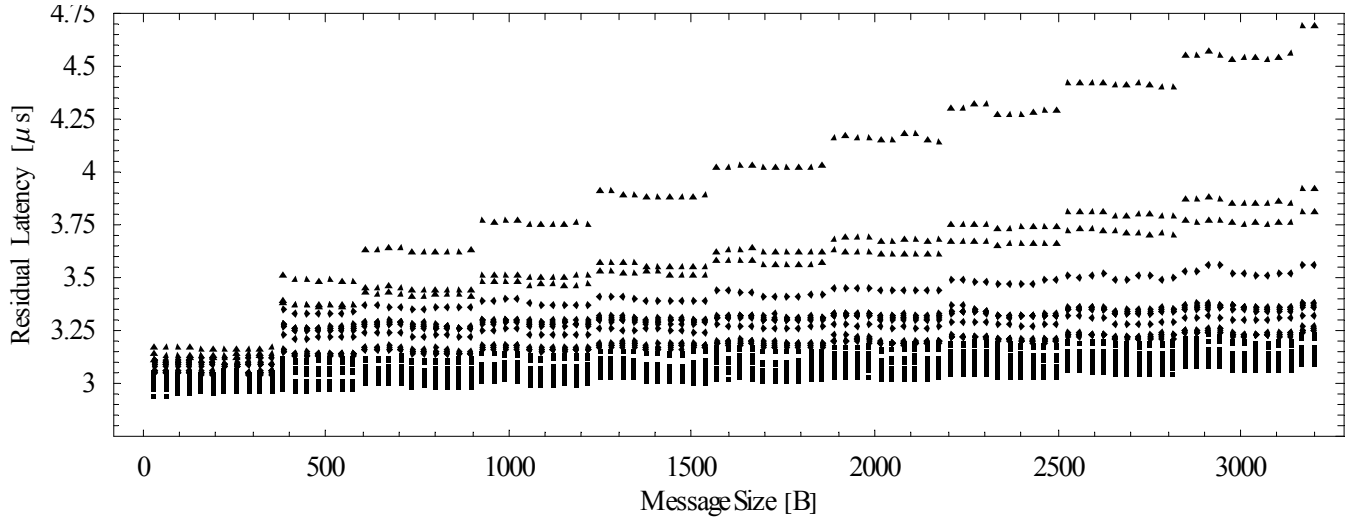

Figure 8: Residual latency (i.e., the measured latency minus the latency associated with the passage of the message and routing data over the network) measured on the *wolverine* machine: The boxes represent messages passing only as far as the first level of the fat-tree network; the diamonds represent messages passing as far as the second level; and the triangles represent messages passing as far as the third level. Each data point represents the average of 200,000 ping measurements.

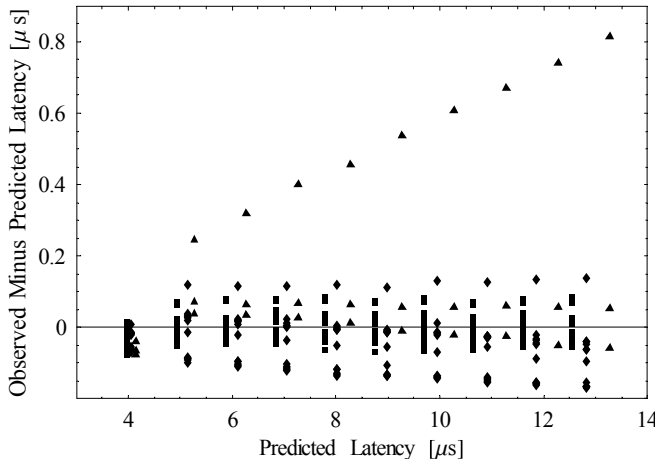

Figure 9: Error in predicting the latency of pings from Elan memory to Elan memory through a network of Elite switches: The boxes represent messages passing only as far as the first level of the fat-tree network; the diamonds represent messages passing as far as the second level; and the triangles represent messages passing as far as the third level. Each data point represents the average of 200,000 ping measurements.

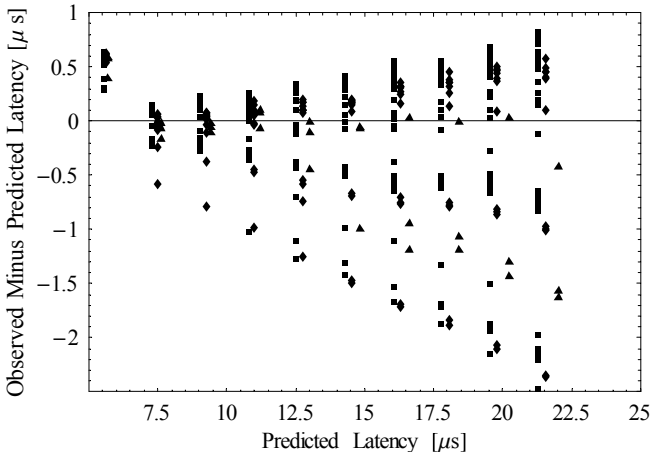

Figure 10: Error in predicting the latency of pings from main memory to main memory through a network of Elite switches: The boxes represent messages passing only as far as the first level of the fat-tree network; the diamonds represent messages passing as far as the second level; and the triangles represent messages passing as far as the third level. Each data point represents the average of 200,000 ping measurements.

[3] Nicol, D.M. and J. Liu. 2002. "Composite Synchronization in Parallel Discrete Event Simulation." *IEEE Transactions on Parallel and Distributed Systems 13,* No. 5, May:433–446.

[4] Cowie, J.H., D.M. Nicol and A.T. Ogielski. 1999. "Modeling the Global Internet." *Computing in Science & Engineering 1,* No. 1, January-February:30–38.

[5] Quadrics Supercomputers World, Ltd. 1999. *Elan Reference Manual.* Bristol, UK, January.

[6] Petrini, F., W. Feng, A. Hoisie, S. Coll and E. Frachtenberg. 2002. "The Quadrics Network: High Performance Clustering Technology." *IEEE Micro 22,* No. 1, January-February:46–57.

[7] Quadrics Supercomputers World, Ltd. 1999. *Elite Reference Manual.* Bristol, UK, November.

[8] `http://www.acl.lanl.gov/30TeraOpRFP/SampleApps/sweep3d/sweep3d.html`.

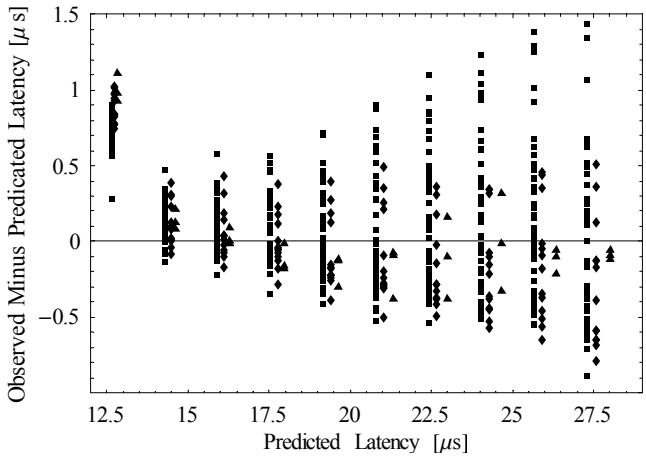

Figure 11: Error in predicting the latency of pings made with MPI through a network of Elite switches: The boxes represent messages passing only as far as the first level of the fat-tree network; the diamonds represent messages passing as far as the second level; and the triangles represent messages passing as far as the third level. Each data point represents the average of 200,000 ping measurements.

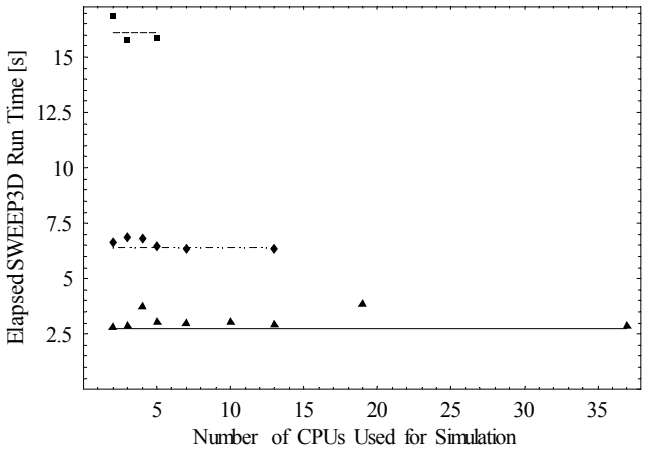

Figure 12: Comparison of SWEEP3D execution time with our simulation of it for three different partitions of a SWEEP3D problem: 2×2 (squares), 3×4 (diamonds), and 6×6 (triangles). The points show the simulation's estimate of the SWEEP3D execution time and the lines show the actual execution time of SWEEP3D for the problem.